Dispute Resolution in Voting

Dispute Resolution in Voting

David Basin
Department of Computer Science
ETH Zurich
basin@inf.ethz.ch

Saša Radomirović
Department of Computer Science
Heriot-Watt University
sasa.radomirovic@hw.ac.uk

Lara Schmid
Department of Computer Science
ETH Zurich
schmidla@inf.ethz.ch

***Abstract*—In voting, disputes arise when a voter claims that the voting authority is dishonest and did not correctly process his ballot while the authority claims to have followed the protocol. A dispute can be resolved if any third party can unambiguously determine who is right. We systematically characterize all relevant disputes for a generic, practically relevant, class of voting protocols. Based on our characterization, we propose a new definition of dispute resolution for voting that accounts for the possibility that both voters and the voting authority can make false claims and that voters may abstain from voting.**

A central aspect of our work is timeliness: a voter should possess the evidence required to resolve disputes no later than the election's end. We characterize what assumptions are necessary and sufficient for timeliness in terms of a communication topology for our voting protocol class. We formalize the dispute resolution properties and communication topologies symbolically. This provides the basis for verification of dispute resolution for a broad class of protocols. To demonstrate the utility of our model, we analyze a mixnet-based voting protocol and prove that it satisfies dispute resolution as well as verifiability and receipt-freeness. To prove our claims, we combine machine-checked proofs with traditional pen-and-paper proofs.

I. Introduction

For a society to accept a voting procedure, the public must believe that the system implementing it works as intended, that is, the system must be *trustworthy*. This is essential as elections involve participants from opposing political parties that may neither trust each other nor the election authority. Nevertheless, there must be a consensus on the final outcome, including whether the election is valid. This requires that voters and auditors can *verify* that the protocol proceeds as specified and detect any manipulations, even if they do not trust the authority running the election. To achieve this, the information relevant for checking verifiability may be published in a publicly accessible database, known as the *bulletin board*.

a) The need for dispute resolution: Ballots are cast privately in elections. Thus only the voters themselves know if and how they voted. If a voter claims that his ballot is incorrectly recorded or that he was hindered in recording his ballot, no other party can know, a priori, whether the voter is lying or if there was a problem for which the voting authority is responsible. We call such unresolved situations *disputes*.

When a dispute occurs, the honest parties must be protected. That is, an honest voter who detects some manipulation must be able to convince third parties that the authority was dishonest.[1] In particular, when a voter checks whether his cast ballot is correctly recorded, then either this is the case (respectively, no ballot is recorded when he abstained from voting) or he can convince others that the authority was dishonest. Another problem is when a voter cannot even proceed in the protocol to perform such checks, for instance when he is not provided with a necessary confirmation. Hence, a timeliness guarantee must ensure by the election's end that an honest voter's ballot is correctly recorded or there is evidence that proves to any third party that the authority is dishonest. Finally, in addition to protecting the honest voters from a dishonest authority, the honest authority must be protected from voters making false accusations. That is, when the authority is honest, no one should be able to convince others of the contrary.

b) State of the art: The vast majority of formal analyses of remote e-voting protocols do not consider dispute resolution at all, e.g., [1], [13], [15], [16]. Works that recognize the importance of dispute resolution [7], [2] or that take aspects of it into account when proposing poll-site [8], [11], [12], [14], [17], [22], [36] or remote [37] voting protocols, reason about it only informally. The most closely related prior works define different notions of *accountability* [10], [25], [26] that formalize which agents should be held accountable when a protocol fails to satisfy some properties. These definitions are very general, but have been instantiated for selected voting protocols [10], [26], [27], [28]. The accountability properties satisfied by these protocols do not guarantee the resolution of all disputes considered by our dispute resolution properties. We provide a detailed comparison of accountability and our properties in Section VII.

c) Contributions: Our work provides a new foundation for characterizing, reasoning about, and establishing dispute resolution in voting. First, we systematically reason about what disputes can arise in voting for a generic, practically relevant, class of voting protocols. Our class comprises both remote and poll-site voting protocols that can be electronic or paper-based. We then focus on disputes regarding whether the published recorded ballots correctly represent the ballots cast by the voters. Based on our classification, we formally define dispute resolution properties in a symbolic formalism amenable to automated verification using the Tamarin tool [30], [34]. This enables the analysis of a broad class

[1]Here dishonesty includes all deviations of the authority from the protocol specification, both due to corruption or to errors.

of protocols with respect to dispute resolution. Moreover, we identify an important new property, which we call *timeliness*, requiring that when a voter's ballot is recorded incorrectly he has convincing evidence of this by the election's end. This property ensures the resolution of disputes that could not be resolved unambiguously in prior work.

Second, we demonstrate that timeliness can only be guaranteed under strong assumptions (for example, some messages must not be lost on the network) by systematically analyzing what communication channels and trust assumptions are necessary and sufficient to satisfy this property. The result is a complete characterization of all topologies in our voting protocol class for which timeliness holds for some protocol. Such a characterization can guide the design of new voting protocols where timeliness should hold, e.g., by identifying and thereby eliminating settings where timeliness is impossible. We formally verify the claimed properties using proofs constructed by Tamarin and pen-and-paper proofs.

Finally, to simplify establishing dispute resolution in practice, we introduce a property, called *Uniqueness*, that can be checked by everyone and guarantees that each recorded ballot was cast by a unique voter. We prove for protocols where voters can cast at most one ballot that *Uniqueness* implies guarantees for voters who abstain from voting. This has the practical consequence that in many protocols, the corresponding guarantees can be proven more easily. We then present as a case study a mixnet-based voting protocol with dispute resolution and prove that our introduced properties hold, as well as standard voting properties such as verifiability and receipt-freeness.

Overall, our results can be used as follows. In addition to specifying what messages are exchanged between the different agents, a voting protocol in our class specifies (i) how the election's result is computed, (ii) which verification steps are performed, and (iii) when the authority conducting the election is considered to have behaved dishonestly. For (i), it is required that each protocol specifies a function *Tally*. For (ii), as voters must be able to check that no ballots were wrongly recorded for them, a function *castBy* must map each ballot to the voter that has (presumably) cast it. Only if this is defined can a voter notice when a ballot was recorded for him that he has not cast. Finally, dispute resolution requires that a protocol defines a dispute resolution procedure such that everyone can agree on (iii). For this purpose, a protocol may specify a set of executions *Faulty* where the authority is considered to have behaved dishonestly and which only depends on public information and can therefore be evaluated by everyone.

Given a protocol with a dispute resolution procedure and a communication topology, our topology characterization can be used to quickly conclude if the given topology is insufficient to achieve the timeliness aspect of dispute resolution. When this is the case, one can immediately conclude that not all dispute resolution properties can be satisfied. When this is not the case, our formal definitions can be used to analyze whether all dispute resolution properties indeed hold in the protocol. Thereby, in protocols where voters can cast at most one ballot, the guarantees for voters who abstain can be established directly or by showing *Uniqueness* and inferring them by our results.

d) Organization: We describe our protocol model in Section II and the class of voting protocols for which we define our properties in Section III. In Section IV, we classify disputes and define our dispute resolution properties. We then analyze in Section V the communication topologies where timeliness can be achieved. In Section VI, we show how dispute resolution can be established in practice, introduce *Uniqueness*, and present our case study. Finally, we discuss related work in Section VII and conclude in Section VIII.

II. Protocol model and system setup

As is standard in model-checking, we model the protocol and adversary as a (global) transition system. Concretely, we use a formalism that also serves as the input language to the Tamarin tool [30]. Our model uses abstractions that ease the specification of communication channels with security properties and trust assumptions. These kinds of abstractions are now fairly standard in protocol specifications. We complement existing abstractions [4] (e.g., authentic and secure channels and parties that satisfy different kinds of trust assumptions) with new abstractions that are relevant for dispute resolution (e.g., reliable channels described in Section II-E2). Our protocol model is inspired by the model in [5] used for e-voting. We first introduce some terminology relevant for voting protocols and then our protocol model.

a) Terminology: We distinguish between *votes* and *ballots*. Whereas a vote is a voter's choice in plain text, a ballot contains the vote and possibly additional information. The ballots' exact design depends on the voting protocol, but it usually consists of the vote cryptographically transformed to ensure the vote's authenticity or confidentiality. When a ballot is sent by the voter, we say it is *cast*. We denote by the *(voting) authority* the entity responsible for collecting and tallying all voters' ballots. Usually, both the list of collected ballots, called the *recorded ballots*, and the *votes in the final tally* are published on a *public bulletin board* that can be accessed by voters and auditors to verify the election's result.

A. Notation and term algebra

We write $[x_i]_{i\in\{1,\dots,n\}}$ to denote a list of n messages of the same kind. Similarly, we write $[f(x_i, y_i)]_{i\in\{1,\dots,n\}}$ for a list whose elements have the same form, but may have different values. When the index set is clear from context, we omit the indices, e.g., we write $[x]$ and $[f(x, y)]$ for the above lists, and we write $[x]_i$ and $[f(x, y)]_i$ for the ith element in the lists. Also, we write $x := y$ for the assignment of y to x.

Our model is based on a term algebra $\mathcal{T}$ that is generated from the application of functions in a signature Σ to a set of names $\mathcal{N}$ and variables $\mathcal{V}$. We use the standard notation and equational theory, given in [4, Appendix A]. The symbols we use here are $\langle p_1, p_2\rangle$ for pairing two terms p_1 and p_2, $\mathit{fst}(p)$ and $\mathit{snd}(p)$ for the first and second projection of the pair p, $\mathit{pk}(x)$ for the public key (or the verification key) associated with a

private key (or signing key) x, and $\{m\}_{sk}$ for a message m signed with the signing key *sk*. The equational theory contains standard equations, for example pairing and projection obey $\mathit{fst}(\langle p_1, p_2\rangle) = p_1$ and $\mathit{snd}(\langle p_1, p_2\rangle) = p_2$. We sometimes omit the brackets $\langle\rangle$ when tupling is clear from the context.

We extend the term algebra from [4] with the following function symbols and equations. We use $\mathit{ver}(s, k)$ for signature verification, where s is a signed message and k the verification key. When the signature in s is verified with the (matching) verification key k, the function returns the underlying signed message m and otherwise it returns a default value $\perp$. This is modeled by the equations $\mathit{ver}(s, k) = m$, if $s = \{m\}_{sk}$ and $k = \mathit{pk}(sk)$, and $\mathit{ver}(s, k) = \perp$ otherwise.

Moreover, we use the function *Tally* to model the tallying process in voting. Given a list of ballots $[b]$, $\mathit{Tally}([b])$ denotes the computation of the votes $[v]$ in the final tally, possibly including pre-processing steps such as filtering out invalid ballots. The exact definition of *Tally* depends on the protocol.

Finally, $\mathit{castBy}(b)$ denotes the voter who is considered to be the sender of a ballot *b*. As with *Tally*, $\mathit{castBy}(b)$ depends on the protocol, in particular on the ballots' design. For example, in a voting protocol where a ballot b contains a voter's identifier (e.g., a code or pseudonym), $\mathit{castBy}(b)$ maps the ballot b to the voter with the identifier included in b. In contrast, in a voting protocol where ballots contain a signature associated with a voter, $\mathit{castBy}(b)$ maps each ballot to the voter associated with the signature contained in b.

As *Tally* and *castBy* are protocol dependent, each concrete protocol specification must define the equations that they satisfy, i.e., extend the term algebra's equational theory with equations characterizing their properties. Note that the functions may not be publicly computable. For example, if only a voter H knows which identifier or signature belongs to him, then other parties are not able to conclude that $\mathit{castBy}(b) = H$.

B. Protocol specification

A protocol consists of multiple *(role) specifications* that define the behavior of the different communicating roles. We model protocols as transition systems that give rise to a trace semantics. Each role specification defines the role's sent and received messages and *signals* that are recorded, ordered sequentially. A signal is a term with a distinguished top-most function symbol. Signals have no effect on a protocol's execution. They merely label events in executions to facilitate specifying the protocol's security properties. We distinguish *explicit signals* that are defined in the specification and *implicit signals* that are recorded during the protocol execution but are not explicitly included in the specification. We explain how we depict protocols as message sequence charts in Section VI-D2.

Roles may possess terms in their *initial knowledge*, which is denoted by the explicit signal *knows*. We require that in a specified role R, any message sent by R must either occur in R's initial knowledge or be deducible from the messages that R initially knows or received in a previous protocol step. Deducibility is defined by the equational theory introduced above. As is standard, whenever R is specified to receive a term that it already has in its knowledge, an agent who instantiates this role will compare the two terms and proceed with the protocol only when they are equal.

C. Adversary model and communication topology

We depict the system setup as a *topology graph* $G = (V, E)$, where the set of vertices V denotes the roles and the set of (directed) edges $E \subseteq (V \times V)$ describes the available communication channels between roles (see e.g. Figure 1). For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, we define the standard subgraph relation $G_1 \subseteq_G G_2$ as $V_1 \subseteq V_2 \wedge E_1 \subseteq E_2$.

By default, we consider a Dolev-Yao adversary [19] who has full control over the network, learns all messages sent over the network, and can construct and send messages herself. Additionally, the adversary can *compromise* participating agents to learn their secrets and control their behavior. In a concrete system model, we limit the adversary by trust and channel assumptions. A *(communication) topology* [4] $T = (V, E, t, c)$ specifies the system setup by a graph $G(T) = (V, E)$, trust assumptions by a function $t : V \mapsto \mathit{trustType}$ mapping vertices to trust types, and channel assumptions by a function $c : E \mapsto \mathit{chanType}$ mapping edges to channel types, which denote a channel with certain properties. The types *trustType* and *chanType* are specified in Section II-E, after the execution model. When c is applied to an edge (A, B), we omit duplicate brackets and write $c(A, B)$ instead of $c((A, B))$.

D. Execution model, signals, properties, and assumptions

During protocol execution, roles are instantiated by agents (i.e., the parties involved in the protocol) and we consider all possible interleavings of agents' runs in parallel with the adversary. A *trace* tr is a finite sequence of multisets of the signals associated with an interleaved execution. We denote by $\mathit{TR}(\mathit{Pr}, T)$ the set of all traces of a protocol *Pr* that is *run in the topology* T, i.e., run in parallel with the adversary defined by the topology T.[2] We write $tr_1 \cdot tr_2$ for the concatenation of two traces tr_1 and tr_2.

As previously explained, a trace may contain implicit signals, which are recorded during execution but omitted from the protocol specification for readability, and explicit signals (containing auxiliary information) that we explicitly add to the protocol specification. Implicitly, when an agent A sends a message m (presumably) to B, the signal $\mathit{send}(A, B, m)$ is recorded in the trace. Similarly, when an agent B receives m (presumably) from A, $\mathit{rec}(A, B, m)$ is recorded. Furthermore, the signal $K(m)$ denotes the adversary's knowledge and is recorded whenever the adversary learns a term m and $\mathit{hon}(A)$ is recorded when an honest agent A instantiates a role.

Furthermore, we use the explicit signal $\mathit{verifyC}(H, b)$ to indicate that an honest agent checks whether the ballot b, which was cast by the honest voter H, is recorded correctly (C stands for *cast* and indicates that H cast a ballot). In protocols where voters can cast multiple ballots, this signal

[2] T may specify channels that are never used by *Pr*. Also, *Pr* may specify channels that are not available in T. In the latter case, the corresponding protocol steps cannot be executed and will not occur in the execution.

can occur multiple times for the same voter. Moreover, the signal *verifyA*(H, b_H) is recorded when an honest agent checks for an honest voter H who has cast the set of ballots b_H, that no ballots other than those in b_H are recorded for H. When this check is done for H who abstained, then $b_H = \emptyset$ (A stands for the fact that H *abstained* from voting). *verifyC* and *verifyA* may be defined such that they can be computed by a machine but not by a human voter, e.g., when they require cryptographic computations. We thus leave it open whether they are performed by the voter H or by another agent such as a helper device.

The explicit signal *knows*(A, x) is recorded when an agent A has a term x in its initial knowledge. The explicit signals *Vote*(H, v) and *Ballot*(H, b) respectively record an honest voter H's vote v and cast ballot b. The former is recorded when H decides what to vote for and the latter is recorded when H casts his ballot. Finally, the explicit signal *BB*(m) denotes that a message m is published on the bulletin board. We use subscripts to distinguish the signals recorded when different messages are published on the bulletin board. For example, the signals $BB_{rec}([b])$ and $BB_{tal}([v])$ denote that the recorded ballots $[b]$ and the votes in the final tally $[v]$ are published. We will introduce further signals as we need them.

We next define two kinds of trace properties. The first are classical *security properties*, which are specified as sets of traces. A protocol *Pr* run in the topology T, satisfies a security property $\mathcal{S}_S$, if *TR*$(Pr, T) \subseteq \mathcal{S}_S$. To reason about functional requirements, we additionally define *functional properties*. For example, the empty protocol satisfies many security properties but it is useless for voting because, even in the absence of the adversary, a voter's ballot is never recorded. We will thus require a functional property stating that a protocol must at least have one execution where a voter's ballot is correctly recorded. We describe a functional property by a set of traces $\mathcal{S}_F$, for example containing all traces where a voter's ballot is recorded. We then define that a protocol *Pr* run in the topology T satisfies the property $\mathcal{S}_F$ if *TR*$(Pr, T) \cap \mathcal{S}_F \neq \emptyset$. Finally, we define protocol *assumptions* as sets of traces. That is, we define so-called *(trace) restrictions* by giving a set of traces and then only consider the traces in the intersection of this set and *TR*(Pr, T) (see e.g. Section II-E2).

E. Trust and channel types

1) Trust types: In the topologies, we consider four types of trust on roles that reflect the honesty of the agents that execute the role. A *trusted* role means we assume that the agents who instantiate this role are always *honest* and thus strictly follow their role specification. In contrast, an *untrusted* role can be instantiated by *dishonest* agents (i.e., compromised by the adversary) who behave arbitrarily. Dishonest agents model both corrupt entities and entities that unintentionally deviate from their specification, for example due to software errors. We model dishonest agents by sending all their secrets to the adversary and by modeling all their incoming and outgoing channels as insecure (see the channel types below).

In addition, we consider the types *trustFwd* and *trustRpl*, which assume *partial trust*. The agents who instantiate a role of type *trustFwd* or *trustRpl* do not strictly follow their role specification but, respectively, always correctly forward messages or reply upon receiving correct messages. Such assumptions turn out to often be necessary for the timeliness property that we introduce shortly, as otherwise dishonest agents that are expected to forward or answer certain messages can fail to do so and thereby block other protocol participants. Thus, these trust types enable fine grained distinctions to be made about which assumptions are necessary for certain properties to hold.

In summary, we consider the set of trust types *trustType* $:= \{$*trusted*, *trustFwd*, *trustRpl*, *untrusted*$\}$. In the topologies, we denote trusted roles by nodes that are circled twice (see e.g., *BB* in Figure 3a, p. 11) and the partial trust types *trustFwd* and *trustRpl* by two dashed circles (see e.g., *P* in Figure 3a). In our protocol class, there is no role that can be mapped both to type *trustFwd* and to type *trustRpl*; thus the interpretation will always be unambiguous. All remaining roles are untrusted.

2) Channel types: In addition to the trust assumptions, a communication topology states channel assumptions. Channels, which are the edges in the topology graph, denote which parties can communicate with each other. Also, channels define assumptions, for example that limit the adversary by stating who can change or learn the messages sent over a given channel. Following Maurer and Schmid [29], we use the notation $A \circ\!\!\rightarrow\!\!\circ\ B$, $A \bullet\!\!\rightarrow\!\!\circ\ B$, $A \circ\!\!\rightarrow\!\!\bullet\ B$, and $A \bullet\!\!\rightarrow\!\!\bullet\ B$ to denote a channel from (instances of) role A to role B that is respectively insecure, authentic, confidential, and secure. For a formal semantics for these channels, see [4].

We introduce two additional channel assumptions that are useful for dispute resolution. These assumptions concern whether a channel reliably delivers messages and whether external observers can see the communication on a channel. Usually, it is assumed that the above channels are *unreliable* in that the adversary can drop messages sent. We make such assumptions explicit and also allow for *reliable* variants of channels. On a reliable channel, the adversary cannot drop messages and thus all messages sent are received by the intended recipient. We will see in Section V that such channels are needed to achieve timeliness properties.

For dispute resolution, it is sometimes required that external observers can witness the communication an agent is involved in to later judge whether this agent is telling the truth. For example, it may be required that witnesses can observe when a voter casts his physical ballot by placing it into a voting box. Such communication cannot later be denied, e.g. when others witness that the voter has cast his ballot then the voter cannot later deny this. Whereas it is in reality sufficient if several witnesses, e.g., a subset of all voters, can observe such communication, we model this by channels that specify that *any* honest agent can observe such communication. Similarly, we will also model the fact that sufficiently many parties can decide who is right in a dispute by specifying that *any* party can resolve disputes (see Section IV). Concretely, we model

communication that can be observed by others by *undeniable* channels where any honest agent $C \notin \{A, B\}$ learns the communication between A and B. This is in contrast to the default *deniable* channels, where an honest agent $C \notin \{A, B\}$ cannot determine that A and B are communicating with each other.

A *channel type* can be built from any combination of the three channel assumptions introduced above. For example, on an insecure reliable channel, the adversary can learn all messages and write messages herself, but she cannot drop messages sent from A to B. However, for dispute resolution not all combinations are useful. In particular, an undeniable channel provides evidence that a message was sent, but this is only useful together with the guarantee that the message is also received. Hence, we only consider undeniable channels that are also reliable. We thus distinguish the following channel types, named after their most significant property: The *default channels* $\circ\xrightarrow{d}\circ$, $\bullet\xrightarrow{d}\circ$, $\circ\xrightarrow{d}\bullet$, $\bullet\xrightarrow{d}\bullet$, which are neither reliable nor undeniable, the *reliable channels* $\circ\xrightarrow{r}\circ$, $\bullet\xrightarrow{r}\circ$, $\circ\xrightarrow{r}\bullet$, $\bullet\xrightarrow{r}\bullet$, which are reliable but not undeniable, and the *undeniable channels* $\circ\xrightarrow{u}\circ$, $\bullet\xrightarrow{u}\circ$, $\circ\xrightarrow{u}\bullet$, $\bullet\xrightarrow{u}\bullet$, which are both reliable and undeniable.

We model the guarantees for senders and receivers that use a reliable or undeniable channel by stating that each message sent on such a channel is also received. We only require this property when both the sender and the receiver of a message are trusted or partially trusted and formally express it by the following restriction.

$$\{tr | \forall A, B, m.\ t(A), t(B) \in \{\mathit{trusted}, \mathit{trustFwd}, \mathit{trustRpl}\} \\ \wedge c(A,B) \in \{\circ\xrightarrow{r}\circ, \bullet\xrightarrow{r}\circ, \circ\xrightarrow{r}\bullet, \bullet\xrightarrow{r}\bullet, \circ\xrightarrow{u}\circ, \bullet\xrightarrow{u}\circ, \circ\xrightarrow{u}\bullet, \bullet\xrightarrow{u}\bullet\} \\ \wedge\ \mathit{send}(A,B,m) \in tr \implies \mathit{rec}(A,B,m) \in tr\}.$$

To model the additional guarantee that undeniable channels provide, additional signals are recorded in the trace when agents communicate over such channels. That is, whenever an agent A sends a message m to B over an undeniable channel, in addition to the signals $\mathit{send}(A,B,m)$ and $\mathit{rec}(A,B,m)$, the signal $\mathit{Pub}(A,B,m)$ is recorded. We formalize this by the following restriction.

$$\{tr | \forall A, B, m.\ c(A,B) \in \{\circ\xrightarrow{u}\circ, \bullet\xrightarrow{u}\circ, \circ\xrightarrow{u}\bullet, \bullet\xrightarrow{u}\bullet\} \wedge \\ (\mathit{send}(A,B,m) \in tr \vee \mathit{rec}(A,B,m) \in tr) \\ \implies \mathit{Pub}(A,B,m) \in tr\}.$$

In the rest of this paper, these two restrictions are always stipulated. That is, whenever we use $\mathit{TR}(\mathit{Pr}, T)$ to refer to all traces of the protocol *Pr* run in the topology T, we actually mean all traces in the intersection of $\mathit{TR}(\mathit{Pr}, T)$ and the above two sets of trace restrictions.

III. Class of voting protocols

Formal reasoning about dispute resolution in voting requires a language for specifying voting protocols and their properties. We provide such a language by presenting a class of voting protocols for which we subsequently define dispute resolution properties. Our class comprises both remote and poll-site voting protocols that can be electronic or paper-based. However, we require a public bulletin board, which is, most of the time, realized by digital means. We define the class by stating natural restrictions that communication topologies and protocols must satisfy to be in our class. Afterwards, we show that many well-known voting protocols belong to this class.

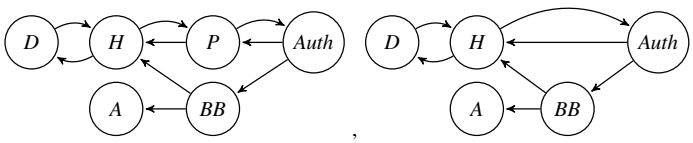


Fig. 1: The topology graphs G_S (left) and G_U (right). We allow for any topology where $G(T) \subseteq_G G_S$ or $G(T) \subseteq_G G_U$.

A. Communication topologies considered

1) Topology graph: The topology graphs in Figure 1 depict all possible roles and communication channels that we consider. That is, we allow for any topology T whose $G(T)$ is a subgraph of G_S or G_U in Figure 1. The node *H* describes two roles for the human voters, one for voters who vote and one for those voters who abstain. Also, there are roles for the devices D and *P*, the voting authority *Auth*, a public bulletin board *BB*, and the auditor *A*. In a concrete protocol, each role, except for *Auth* and *BB*, can be instantiated by multiple agents.

We consider two kinds of setups, G_S and G_U in Figure 1, for two kinds of protocols that differ in how ballots are cast. G_S provides the necessary channels for protocols where each voter H knows his ballot and sends it to *Auth* using a platform *P*. It models remote and poll-site voting. In the former case *P* could be the voter's personal computer, and in the latter case *P* could be a ballot box, or an optical scanner that forwards H's ballot b to the authority for tallying. G_U models setups for protocols where a trusted platform P computes (e.g., encrypts) and casts the ballot for H. Often, such protocols do not distinguish between H and P and P operates "in the name of H". Therefore, we model the setup of such protocols by unifying the roles H and P into a single role *voter* H.

In some protocols, voters also have a personal off-line device *D*. In contrast to *P*, *D* has limited capabilities and is not connected to the authority. This models, for example, off-line trusted digital devices or letters containing codes that may be used to compute ballots.

Auth denotes the authority that is responsible for setting up elections and collecting and tallying the ballots. Even though some voting protocols describe the authority in terms of several distinct roles, we collectively describe all these relevant functionalities in a single role, except for the publication of information, which is described by the bulletin board role *BB*. We then also consider just one agent in the role *Auth*. We will argue in Section IV that this is sufficient for our dispute resolution properties. As depicted in Figure 1, *Auth* can publish information on *BB*, which can be read by the auditors and voters. An auditor performs checks on the published

information to ensure that the election proceeded correctly. By modeling the auditor as a separate role, this role can be instantiated by anyone, including the voters.

2) Topology assumptions: We further restrict the considered communication topologies by making some minimal channel and trust assumptions. As is common for many voting protocols [5], [15], [16], [21], we model a secure bulletin board and consider its realization as a separate problem. Such a bulletin board can be used to send messages authentically and consistently from the authority to all voters and auditors. We thus assume that the roles *BB* and A exist and are trusted and that the channels from *Auth* to *BB* as well as from *BB* to *H* and *A* exist and are default authentic channels. Furthermore, we only use the following partial trust assumptions. *Auth* can be trusted to always reply with a confirmation upon receiving a correct message (type *trustRpl*) and *P* can be trusted to always forward messages correctly (type *trustFwd*), e.g., a voting machine can be trusted to forward the entered ballots to a remote server. The remaining channel and trust types can respectively be assigned to any channels and roles. Note that we support protocols using anonymous channels (e.g. Civitas [15]) since, for the properties we consider, anonymous channels can be modeled as our default channels.[3]

For dispute resolution, certain guarantees should hold for an honest voter H, even with an untrusted or partially trusted authority and even if all other voters are untrusted. Similarly, the guarantees for the honest authority should hold even when all voters are untrusted. We therefore only consider topologies T where the roles *H* and *Auth* are untrusted or partially trusted and analyze dispute resolution with respect to three variations of T. We introduce the following notation. We single out a distinguished voter H for whom the security properties are analyzed. Given a topology T, $T^{Auth^+H^+}$ denotes the same topology but where the trust assumptions about *Auth* and the distinguished H are defined by $t(Auth) = trusted$ and $t(H) = trusted$, T^{Auth^+} is as T but with $t(Auth) = trusted$, and T^{H^+} is as T but with $t(H) = trusted$. Note that in all variations, the trust assumptions about the voters other than H are as in T.

B. Voting protocols considered

We next define the voting protocols considered in terms of the protocols' structure and which equations must be specified in the term algebra. Our definition allows for protocols with re-voting, that is, where voters can send several ballots (e.g., [15]). As explained in Section III-A, we allow protocols where the voter H knows and casts his ballot or where a trusted platform P casts the ballot, in which case we unify the roles H and P.

1) Required functions and equations: A protocol specification must define the equation satisfied by $Tally([b])$, defining how the election's result is computed from the list of recorded ballots $[b]$. Similarly, a protocol must define the equation satisfied by $castBy(b)$, which must map each ballot b to a voter, thereby specifying that this voter is considered to have cast the ballot.

2) Protocol's start and end: We assume that the protocol's setup can specify any number of voters and devices and any relation between them, for example that each voter is associated with a unique trusted device. Also, at the protocol's start some public information may be posted on the bulletin board. For example, this might be some election parameters or the list of all eligible voters, denoted by $BB_H([H])$. Furthermore, some agents may know some terms such that these terms (or associated terms) are initially published on the bulletin board or known to other agents. For example, *Auth*'s public key $pk_{Auth} := pk(sk_{Auth})$ can be posted on the bulletin board at the protocol's start whereby *Auth* has the corresponding private key sk_{Auth} in its initial knowledge. We require, however, that at the protocol's start no honest agent knows a voter H's ballot other than the voter himself.

We also assume that an election has two publicly known deadlines that determine the *voting phase's end*, i.e., when ballots can no longer be cast, and the *moment when all relevant information is published*. We denote the latter by the explicit signal *End* in the *BB* role, which is recorded right after the last message relevant for the election is published.

3) Tallying and publication of results: We assume that ballots are collected and tallied by the authority *Auth* and that the protocol allows voters to abstain from voting. Thus, *Auth* starts the tallying process after the voting phase, even if not all voters have cast a ballot. By the election's end, all valid ballots that were received by the authority have been published on the bulletin board together with the votes in the final tally. In protocols with *re-voting*, all ballots are published in the list of recorded ballots and the tallying process is responsible for removing duplicates. Finally, we assume that all messages sent to the bulletin board are immediately published. That is, whenever *BB* receives a message m, the signal $BB(m)$ is recorded in the trace.

C. Examples of protocols in our voting class

Our class comprises well known voting protocols such as Helios [1], Belenios [16], and Civitas [15]. In these protocols, voters can abstain from voting, the bulletin board is assumed to be secure, and the recorded ballots are published on the bulletin board as they were received by the authority. Moreover, even though these protocols all specify different roles and setups, they can each be understood as instantiations of the setups in Figure 1. Belenios and Civitas both have many authority roles, such as registrars and different trustees, which can be understood as our role *Auth*. In Belenios, the *Bulletin Board* also performs some checks and computations. Thus, to cast it in our protocol class, we must additionally interpret those parts of Belenios's *Bulletin Board* as part of our role *Auth* and just the published part of Belenios's *Bulletin Board* as our role *bulletin board BB*.

Note that there are voting protocols, such as BeleniosRF [13], where the recorded ballots are re-encrypted be-

[3]Distinguishing between anonymous and default channels is relevant when analyzing observational equivalence properties such as coercion resistance. However, for our possibility results, we only consider reachability properties.

fore being published on the bulletin board to achieve stronger privacy properties. Such protocols are not in our class.

IV. Dispute Resolution

In voting, the authority conducting the election should behave as expected. That is, if the authority is dishonest, it must be held accountable for this. For elections that are conducted by multiple parties, we require that it is unambiguously detectable when *any* of these parties misbehave, but we do not require that it is detectable *which* of these parties misbehave. This is sufficient to determine when "the system" running the election does not proceed as expected and to take recovery measures when this is the case, such as declaring the election to be invalid. Thus, except for the bulletin board, we model all of the parties involved in conducting an election as one role (and agent) *authority Auth* and require that this agent is held accountable if it does not behave as expected, i.e. does not follow its role specification.

In contrast to the authority, we should not and cannot require that all other parties, notably the voters, behave as expected. In fact, a well-designed voting protocol should still satisfy its expected properties for the honest voters, even when other voters misbehave. We therefore only consider disputes with respect to claims that the voting authority is dishonest.

We first explain why dispute resolution is needed in elections and characterize all relevant disputes. Afterwards, to formalize our dispute resolution properties, we extend our protocol model with additional signals and functions. We then motivate the required properties using our classification and formalize them using the model extensions.

A. Relevant disputes

After an election, all honest participants should agree on the election's outcome. A protocol where any manipulation by the authority can be detected by suitable checks is called *verifiable*. For voting, the gold standard is *end-to-end verifiability* where the final tally consists of the honest voters' votes, tallied correctly and this can all be checked. Often, this property is divided into *universal* and *individual* verifiability. *Universal verifiability* properties denote that some guarantees hold (e.g., the tally is computed correctly) if an auditor performs appropriate checks on bulletin-board data. Any voter or independent third party can serve as an auditor and do such checks. Therefore, if the universal verifiability checks fail, all honest protocol participants will agree on this fact and such checks never give rise to disputes.

Individual verifiability denotes that each voter can verify that his own ballot has been correctly considered in the list of recorded ballots. As only the voter knows which ballot he has cast, this property relies on checks that must (and can only) be done by *each voter himself*. Hence, individual verifiability checks give rise to the following three problems, where a voter claims that the authority is dishonest while other protocol participants cannot determine whether the voter is lying.

(1) A voter is hindered from taking the protocol step where he casts his ballot, in particular he cannot complete one of the preceding protocols steps. There may be technical as well as social reasons for this. For example, the voter may fail to be provided with the necessary credentials in a setup phase or he cannot access a polling station. For generality, we therefore consider disputes regarding the inability to cast a ballot as out of scope of this paper and focus in the following on disputes concerning whether the recorded ballots correctly reflect the ballots cast by the voters.

(2) A voter who successfully cast his ballot is hindered from reaching the verifiability step. For instance, this can happen when recording a ballot requires receiving a confirmation from the authority, which is sent to the voter too late or not at all.

(3) A voter's check whether his ballot is recorded correctly fails. This can happen when a voter detects that one of his cast ballots was not recorded correctly or when he detects that there is a ballot recorded for him that he never sent.

As a result of the above reasoning, based on (3) we distinguish two possible disputes that must be considered in voting protocols, which are depicted in Figure 2. In both disputes, a voter H's and the authority's claim about H's cast ballot differ, where the authority's claim is denoted by the information on the bulletin board. We take the standpoint that the authority is responsible for setting up a working channel to the bulletin board. That is, if messages are not on the bulletin board that should be there, we consider this to be the authority's fault. In the dispute *D1*, a voter H claims that he cast a ballot b, while the authority *Auth* claims that H did not cast b and in the dispute *D2* their claims are reversed. Note that when H claims to have cast b and *Auth* claims that H cast b', this constitutes both a dispute *D1* with respect to the ballot b and a dispute *D2* with respect to b'.

We require that when a voter learns that the authority *Auth* did not record a ballot that he cast, he can convince the other honest participants that *Auth* is dishonest. This is a prerequisite needed to take recovery measures when such manipulations occur. The same must hold when a voter learns that *Auth* recorded a ballot for him that he did not cast. We respectively denote these properties by $\mathit{VoterC}(\mathit{Auth})$ (in dispute *D1*) and $\mathit{VoterA}(\mathit{Auth})$ (in dispute *D2*).

As explained in (2), it is also a problem when a voter who casts a ballot is hindered from reaching his verifiability check in due time. We thus require that a voter who casts a ballot has some timeliness guarantees, namely that by the election's end either his ballot is correctly recorded or he has evidence to convince others that the authority *Auth* is dishonest. We denote this property by $\mathit{TimelyP}(\mathit{Auth})$.

Finally, it is possible that voters lie. Therefore, we require that an honest authority *Auth* is protected from false convictions in any dispute. We denote this by $\mathit{AuthP}(\mathit{Auth})$.

Some protocols support re-voting, where voters can send a set of ballots, all of which are recorded on the bulletin board. In this case, the dispute *D1* denotes that H claims that at least one of his cast ballots is not listed by *Auth*. We thus require that $\mathit{VoterC}(\mathit{Auth})$ and $\mathit{TimelyP}(\mathit{Auth})$ hold for each cast ballot. Dispute *D2* means that H claims not to have cast some of the ballots that are recorded for him. In such a dispute, the

	Voter H claims that H	**Authority** *Auth* claims that H	Properties protecting H	Properties protecting *Auth*
D1	***cast*** *ballot b*	***did not cast*** *ballot b*	*VoterC*(*Auth*), *TimelyP*(*Auth*)	*AuthP*(*Auth*)
D2	***did not cast*** *ballot b*	***cast*** *ballot b*	*VoterA*(*Auth*)	*AuthP*(*Auth*)

Fig. 2: Possible disputes in voting. The authority's claim is captured by the information on the bulletin board. The respective disputes can be resolved when all properties in the third and fourth columns hold.

voter must be able to convince everyone that too many ballots are recorded for him and that the authority *Auth* is dishonest. This guarantee generalizes the property *VoterA*(*Auth*), which we will define so that it covers both situations. As before, the disputes *D1* and *D2* can occur simultaneously, for example when H claims he cast the ballots b_1 and b_2 and *Auth* claims that H cast b_2 and b_3.

B. Protocol model for dispute resolution

To formalize dispute resolution for our class of voting protocols, we extend our protocol model from Section II.

It may be required that agents collect evidence to be used in disputes. We use the signal $Ev(b, ev)$ to model that the evidence *ev* concerning the ballot *b* is collected. We model the *forgery of such evidence* by allowing any dishonest agent to claim that any term in its knowledge is evidence. That is, we allow the adversary to perform an action that records $Ev(b, ev)$ for any terms *b* and *ev* such that $K(\langle b, ev\rangle)$.

As we have argued that all honest agents should be able to agree on the outcome of disputes, we do not specify which agents resolve disputes and how the collected evidence must be communicated to them to file disputes. We merely define that a voting protocol can generate evidence such that *any* third party who obtains this evidence can, together with public information, judge whether the authority *Auth* is dishonest. Recall that in poll-site settings, undeniable channels are used to model that *sufficiently many* witnesses can observe the relevant communication in practice. In this context, requiring that any third party can judge whether the authority is dishonest models that sufficiently many parties can decide this in practice. Abstracting away from these details allows us to focus on which evidence and observations are required to resolve disputes, independently of how undeniable channels are realized in practice.

We thus model the verdict of whether *Auth* should be considered dishonest by a publicly verifiable property *Faulty*, which can be specified as part of each voting protocol, independently of any role. *Faulty*$(Auth, b)$ defines a set of traces where the agent *Auth* is considered to have behaved dishonestly with respect to the ballot *b*, i.e., *b* has presumably not been processed according to the protocol in these traces. For example, *Faulty*$(Auth, b) := \{tr | \exists B, [b].\ Pub(Auth, B, b) \wedge BB_{rec}([b]) \in tr \wedge b \notin [b]\}$ specifies that *Auth* is considered dishonest in all traces where the ballot *b* was sent from *Auth* to an agent B on an undeniable channel but not included in the recorded ballots $[b]$ published on the bulletin board.

To ensure that the verdict whether a trace is in the set *Faulty*$(Auth, b)$ is publicly verifiable, the specification of *Faulty*$(Auth, b)$ must depend just on evidence and public information. We thus state the following requirement.

Requirement 1. *Faulty*$(Auth, b)$ may only be defined based on the signals *BB*, *Ev*, and *Pub*.

Whereas the above example satisfies this requirement, the set $\{tr | \exists A, B, m, [b].\ send(A, B, m) \wedge BB_{rec}([b]) \in tr \wedge m \notin [b]\}$ is not a valid definition of *Faulty*$(Auth, b)$, as *send* is not one of the admissible signals.

As a consequence of the above requirement, not all signals in a trace tr are relevant for evaluating whether tr satisfies a given *Faulty* definition. In particular, let $pubtr(tr)$ be a projection that maps a trace tr to the signals in tr whose top-most function symbol is one of *BB*, *Ev*, or *Pub*, while maintaining the order of these signals. Then, it follows from Requirement 1 that for all traces tr_1 and tr_2 such that $pubtr(tr_1) = pubtr(tr_2)$, it holds that $tr_1 \in$ *Faulty*$(Auth, b)$ iff $tr_2 \in$ *Faulty*$(Auth, b)$.

C. Formal dispute resolution properties

We now use our extended model to define the dispute resolution properties for our class of voting protocols. We formalize each property from Figure 2 as a set of traces.

First, we consider the property *VoterC*$(Auth)$ that protects an honest voter who detects that one of his cast ballots is not recorded correctly by the authority *Auth*. Intuitively, we require that if this happens, the voter can then convince others that the authority is dishonest. Specifically, the property states that whenever an honest voter H (or one of his devices) reaches the step where he believes that one of his ballots *b* should be recorded on *BB*, then either this ballot is correctly included in the list of recorded ballots on *BB* or everyone can conclude that the authority *Auth* is dishonest. We define *VoterC*$(Auth)$ as follows (C denotes that a ballot has been cast).

Definition 1.

$$\begin{aligned} &\textit{VoterC}(Auth) := \{tr \mid \textit{verifyC}(H, b) \in tr \\ &\implies (\exists [b].\ BB_{rec}([b]) \in tr \wedge b \in [b]) \vee tr \in \textit{Faulty}(Auth, b)\}. \end{aligned}$$

Note that, for notational simplicity, here and in the rest of the paper, when using set comprehension notation like $\{x | F(x, \bar{y})\}$, all free variables $\bar{y}$ different from x are universally quantified, i.e., $\{x | \forall \bar{y}.\ F(x, \bar{y})\}$.

The next property, *TimelyP*$(Auth)$, states that an honest voter H who casts a ballot *b* cannot be prevented from proceeding in the protocol such that his ballot is recorded or, if he is prevented, then he can convince others that the authority *Auth* is dishonest. In particular, a voter's ballot must be recorded or there must exist evidence that the authority is dishonest

within a useful time period. Note that we do not require that the resolution of disputes must take place before the election's end and there can be a complaint period afterwards. However, we require that the necessary evidence exists by this fixed deadline as otherwise it could be received after the complaint period ended. We now define *TimelyP*(*Auth*).

Definition 2.

$$\begin{aligned}\textit{TimelyP}(\textit{Auth}) := \{tr \mid \exists tr', tr''.\ tr = tr' \cdot tr'' \\ \wedge \textit{Ballot}(H, b) \in tr' \wedge\ \textit{End} \in tr'' \\ \implies (\exists [b].\textit{BB}_{rec}([b]) \in tr' \wedge b \in [b]) \vee tr \in \textit{Faulty}(\textit{Auth}, b)\}.\end{aligned}$$

The difference to *VoterC*(*Auth*) (Definition 1) is that we not only require the property when a verifiability check is reached, but whenever all the relevant information is published on the bulletin board (indicated by *End*) and a voter's ballot was cast before this deadline. We illustrate this difference on an example in Section VI-D3.

For abstaining voters we define *VoterA*(*Auth*). It states that when an honest voter H who abstains from voting, or one of H's devices, checks that no ballot is recorded for H, then either this is the case or everyone can be convinced that the authority *Auth* is dishonest. We define this property such that it can also be used in protocols with re-voting, where a voter who cast a set of ballots b_H checks that no additional ballots are wrongly recorded for him. We define *VoterA*(*Auth*) as follows.

Definition 3.

$$\begin{aligned}\textit{VoterA}(\textit{Auth}) := \{tr \mid \textit{verifyA}(H, b_H) \in tr \\ \implies \neg(\exists [b], b.\ \textit{BB}_{rec}([b]) \in tr \wedge b \in [b] \wedge\ \textit{castBy}(b) = H \\ \wedge\ b \notin b_H) \vee\ \exists b.\ tr \in \textit{Faulty}(\textit{Auth}, b)\}.\end{aligned}$$

Note that *castBy* is just a claim that H has cast a ballot and does not imply that H has actually cast it. For example, in a protocol where ballots contain a voter's identity in plain text and *castBy*(b) is defined to map each ballot to the voter whose identity it contains, everyone can construct a ballot b such that $H = \textit{castBy}(b)$, even when H has not cast it.

It is possible, of course, that a voter who claims that the authority is dishonest is lying. Thus, for dispute resolution to be fair, it must not only protect the honest voters but also an honest authority. We formalize by *AuthP*(*Auth*) that traces where the authority *Auth* is honest should not be in *Faulty*(*Auth*, b) for any ballot b.

Definition 4.

$$\begin{aligned}\textit{AuthP}(\textit{Auth}) := \{tr \mid \textit{hon}(\textit{Auth}) \in tr \\ \implies \forall b.\ tr \notin \textit{Faulty}(\textit{Auth}, b)\}.\end{aligned}$$

Even though the above properties are stated independently of any adversary model, *VoterC*(*Auth*), *TimelyP*(*Auth*), and *VoterA*(*Auth*) are guarantees for an honest voter *H* and must hold even when the authority *Auth* and other voters are dishonest. Similarly, *AuthP*(*Auth*) constitutes a guarantee for *Auth* and must hold even if all voters are dishonest. Therefore, we define a *dispute resolution property* by stating what property must be satisfied by a protocol (1) for the honest voter H, i.e., when the protocol is run in a topology where H is honest, and (2) for the honest authority *Auth*. Additionally, it is usually required that a protocol satisfies some functional requirement when the agents are honest. Thus a dispute resolution property also specifies (3) which functional requirement must hold when both *Auth* and the voter H are honest.

Definition 5. Let p_H and p_{Auth} be two security properties that must hold respectively for an honest voter H and the honest authority *Auth* and let p_f be a functional property that must hold when both agents are honest. A protocol *Pr*, executed in a topology T, satisfies the dispute resolution property $\textit{DR}(\textit{Pr}, T, p_H, p_{Auth}, p_f)$ iff

$$\begin{aligned}&\textit{TR}(\textit{Pr}, T^{\textit{Auth}^+ H^+}) \subseteq p_H \cap p_{Auth} \wedge \textit{TR}(\textit{Pr}, T^{H^+}) \subseteq p_H \\ &\wedge \textit{TR}(\textit{Pr}, T^{\textit{Auth}^+}) \subseteq p_{Auth} \wedge \textit{TR}(\textit{Pr}, T^{\textit{Auth}^+ H^+}) \cap p_f \neq \emptyset.\end{aligned}$$

For example, $\textit{DR}(\textit{Pr}, T, \textit{VoterC}(\textit{Auth}) \cap \textit{TimelyP}(\textit{Auth}) \cap \textit{VoterA}(\textit{Auth}), \textit{AuthP}(\textit{Auth}), f)$ denotes that the protocol *Pr* run in the topology T satisfies all previously introduced properties in the required adversary models. That is, it satisfies the properties *VoterC*(*Auth*), *TimelyP*(*Auth*), and *VoterA*(*Auth*) for an honest voter H, the property *AuthP*(*Auth*) for an honest authority *Auth*, and the functional property f for an honest voter and the honest authority (see the next section for an example of a functional property). Another dispute resolution property that we consider in the next Section is $\textit{DR}(\textit{Pr}, T, \textit{TimelyP}(\textit{Auth}), \textit{AuthP}(\textit{Auth}), f)$, which states that the timeliness property *TimelyP*(*Auth*) should hold for an honest voter H while *AuthP*(*Auth*) is preserved for the honest authority *Auth*.

V. Communication topologies and timeliness

For *TimelyP*(*Auth*), it is a problem when messages are lost as some protocol participants may be waiting for these messages and thus cannot proceed in the protocol. Intuitively, timeliness only holds under strong assumptions. We investigate this next by systematically characterizing the assumptions needed for *TimelyP*(*Auth*) to hold in our protocol class.

A. Problem scope

1) Dispute resolution property: We aim at achieving timeliness guarantees for the voters while also maintaining the *AuthP*(*Auth*) property for the authority *Auth*. Furthermore, we are only interested in protocols where a voter's ballot can actually be recorded. To express the third requirement, we formalize a functional property stating that for a given protocol and topology, there must be an execution where an honest voter *H* casts a ballot b and where this ballot is published in the list of recorded ballots $[b]$ on the bulletin board before the last relevant information is published (indicated by *End*). Moreover, this property must hold when all agents and the network behave honestly. We denote by *honestNetw* the set of traces where all agents follow the protocol and messages are forwarded on all channels unchanged. The required functional property is defined as the following set of traces.

Definition 6.

$$\begin{aligned}\mathit{Func} \;:=\; \{tr \mid \exists tr', tr'', H, b, [b].\; tr = tr' \cdot tr'' \wedge \\ \mathit{Ballot}(H, b) \in tr' \wedge \mathit{BB}_{\mathit{rec}}([b]) \in tr' \wedge b \in [b] \wedge \mathit{End} \in tr'' \\ \wedge\; tr \in \mathit{honestNetw}\}.\end{aligned}$$

Given a protocol *Pr* and a topology T, we would like the dispute resolution property *DR*(Pr, T, *TimelyP*(*Auth*), *AuthP*(*Auth*), *Func*), which we write for conciseness as *TimelyDR*(Pr, T).

2) Additional assumptions: In standard protocol models, honest agents can stop executing their role at any time. For timeliness, this might be a problem as other agents may wait for their messages and cannot proceed in the protocol. We thus state the additional assumption that honest agents do not abort the protocol execution prematurely. Similarly, we assume that partially trusted agents execute the required action once they can. Note that agents can still be blocked, e.g., when waiting for messages that are dropped on the network.

Assumption 1. Honest agents always execute all protocol steps possible and agents who instantiate a role that is trusted to forward or answer messages, i.e., partially trusted, perform this respective action once they can.

The assumption implies, for example, that when a role specifies a send event after a receive event, the agent instantiating the role will always perform the second step right after the first one. However, an agent can also be blocked between two consecutive protocol steps, for example when multiple receive events are specified after each other and the agent must wait for all of them.

Under the above assumption, we characterize all topologies from our protocol class for which there exists a protocol that satisfies *TimelyDR*, i.e., the set $\{T | \exists Pr.\ \mathit{TimelyDR}(Pr, T)\}$. As others [4], we introduce a partial order on topologies such that a possibility result (i.e., the existence of a protocol such that *TimelyDR*(Pr, T) holds) for a weaker topology implies a possibility result for a stronger topology. We then characterize the above set by providing the "boundary" topologies, i.e., the minimal topologies satisfying *TimelyDR*.

B. Communication topology hierarchy

We define a partial order on topologies that, given two topologies, orders them with respect to their trust and system assumptions. We first define a partial order on our trust and channel types. For $t, t' \in \mathit{trustType}$, $t' \sqsubseteq t$ denotes that t is a stronger assumption than t'. We thus have that *untrusted* $\sqsubseteq$ *trustFwd* $\sqsubseteq$ *trusted* and *untrusted* $\sqsubseteq$ *trustRpl* $\sqsubseteq$ *trusted*. We also define for two channel types c and c' that c makes stronger assumptions than c'. Formally, let $\sqsubseteq_0$ be the relation where $\circ \xrightarrow{x} \circ \sqsubseteq_0 \circ \xrightarrow{x} \bullet \sqsubseteq_0 \bullet \xrightarrow{x} \bullet$ and $\circ \xrightarrow{x} \circ \sqsubseteq_0 \bullet \xrightarrow{x} \circ \sqsubseteq_0 \bullet \xrightarrow{x} \bullet$ for all $x \in \{d, r, u\}$ and $\xrightarrow{d} \sqsubseteq_0 \xrightarrow{r} \sqsubseteq_0 \xrightarrow{u}$ for all $\rightarrow \in \{\circ\rightarrow\circ, \bullet\rightarrow\circ, \circ\rightarrow\bullet, \bullet\rightarrow\bullet\}$. We overload the symbol $\sqsubseteq$ and, for channel types, we write $\sqsubseteq = \sqsubseteq_0^*$, i.e., $\sqsubseteq$ is the reflexive transitive closure of $\sqsubseteq_0$.

Using the above, for two topologies $T_1 = (V_1, E_1, t_1, c_1)$ and $T_2 = (V_2, E_2, t_2, c_2)$ we say that T_2 makes at least as strong assumptions as T_1 if T_2 uses channel and trust assumptions that are at least as strong as those in T_1 and if T_2's topology graph includes all the roles and communication channels that exist in T_1:

$$\begin{aligned}T_1 \sqsubseteq T_2 := G(T_1) \subseteq_G G(T_2) \wedge \forall (v_a, v_b) \in E_1. \\ c_1(v_a, v_b) \sqsubseteq c_2(v_a, v_b) \wedge \forall v \in V_1.\; t_1(v) \sqsubseteq t_2(v).\end{aligned}$$

We show next that defining the relation this way is useful for relating possibility results for different topologies. In particular, if for a topology T it is possible to satisfy *TimelyDR*(Pr, T) with some protocol, then for all topologies that make stronger assumptions, there is also a protocol that satisfies the property. The lemma is proven in Appendix A1 of the full version of the paper [6].

Lemma 1. Let $T_I \sqsubseteq T_S$ be topologies in our class.

$$\exists Pr.\ \mathit{TimelyDR}(Pr, T_I) \implies \exists Pr'.\ \mathit{TimelyDR}(Pr', T_S).$$

C. Characterization of topologies enabling TimelyDR

We next present the minimal topologies satisfying *TimelyDR* in our voting protocol class. In combination with the above hierarchy, this allows us to fully characterize all topologies T that enable *TimelyDR*(T, Pr) for some protocol *Pr*.

The minimal topologies are depicted in Figure 3 and denoted by $T_1, \ldots, T_7$. Recall that the agents *BB* and *A* as well as their incoming and outgoing channels have fixed trust assumptions. In all topologies, there are roles for *H*, *P* and *Auth* (respectively for *H* and *Auth* in T_6 and T_7), as this is required to satisfy the functional property (*H* must cast a ballot, *P* must forward it, and *Auth* must publish it on *BB*). All topologies have a reliable path from *H* to *Auth* and additional trust assumptions, such as (partially) trusted roles or undeniable channels. We present some possible real-world interpretations of these topologies in Section VI-A.

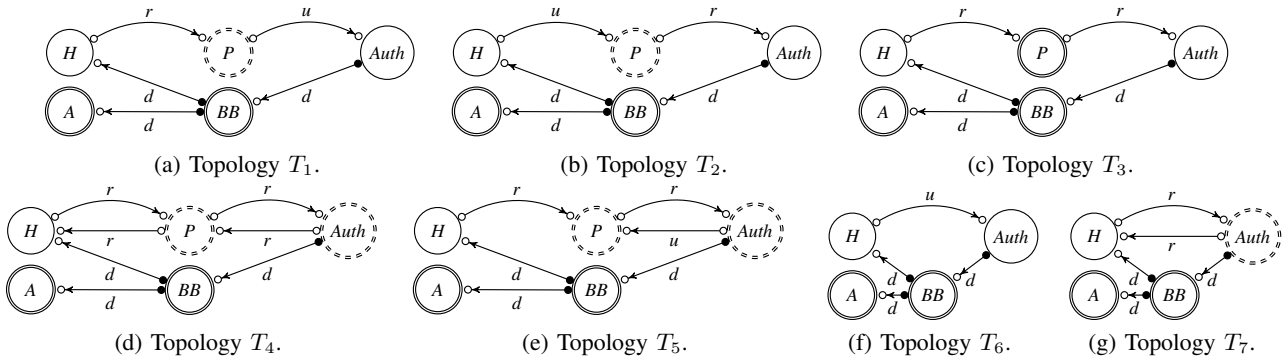
We now state the main theorem for our voting protocol class: The set of topologies for which there exists a protocol that establishes *TimelyDR* consists of all topologies that make at least the assumptions that are made by one of the seven topologies in Figure 3.

Theorem 1. Let $T_1, \ldots, T_7$ be the topologies depicted in Figure 3 and T be a topology in our voting protocol class.

$$(\exists Pr.\ \mathit{TimelyDR}(Pr, T)) \Leftrightarrow (\exists i \in \{1, \ldots, 7\}.\; T_i \sqsubseteq T).$$

We only explain the high level idea of the proof here and refer to [6, Appendix A2] for the details.

Proof Sketch. First, we establish necessary requirements for topologies to enable *TimelyDR*, by showing by pen-and-paper proofs that any topologies that do not meet these requirements cannot satisfy *TimelyDR* with any protocol. Next, we show that these requirements, which are met by the topologies $T_1, \ldots, T_7$ in Figure 3 are sufficient. In particular, we prove by automated proofs in Tamarin (see [33]) that for each topology $T_i \in \{T_1, \ldots, T_7\}$ there exists a simple protocol Pr_i for which *TimelyDR*(T_i, Pr_i). Finally, we show (by hand) that the topologies $T_1, \ldots, T_7$ in Figure 3 are the only *minimal* topologies satisfying the necessary and sufficient requirements.



(a) Topology T_1. (b) Topology T_2. (c) Topology T_3.

(d) Topology T_4. (e) Topology T_5. (f) Topology T_6. (g) Topology T_7.

Fig. 3: The minimal topologies for which there exists a protocol such that *TimelyDR* can be achieved. The channels' labels denote whether the channels are default (d), reliable (r), or undeniable (u). The nodes' lines denote whether the roles are untrusted (circled once), trusted (circled twice), or partially trusted (dashed circles), where a partially trusted P is of type *trustFwd* and a partially trusted *Auth* is of type *trustRpl*.

It follows that all topologies in our class are either stronger than one of the topologies $T_1, \ldots, T_7$ and, by Lemma 1, also also establish *TimelyDR* with some protocol or they are weaker than one of the topologies $T_1, \ldots, T_7$ and thus do not meet the necessary requirements for *TimelyDR*. □

The theorem shows that strong assumptions are indeed necessary to achieve timeliness for dispute resolution. In particular, unreliable channels are insufficient. In most cases, undeniable channels or trusted platforms are required. This can only be avoided in those topologies where there are reliable paths both from the voters to the authority and back. Moreover, *TimelyDR* cannot hold when the platforms are untrusted. This generalizes [2], which states that dispute resolution (called *contestability* in [2]) cannot hold in poll-site voting protocols where ballot-marking devices can be corrupted.

Recall that, in our protocol class, we also allow for off-line devices *D*. Our analysis shows that *D* is irrelevant for the question of whether or not *TimelyDR* can be achieved. Also, it is irrelevant whether the channels between the voters and the authority are authentic, confidential, or secure. In particular, they can all be insecure. Nevertheless such devices and channels are needed in voting to satisfy other properties, for example privacy.

VI. Dispute resolution in practice

We now give a practical interpretation of the above results and illustrate how our formalism can be used.

A. Topologies providing TimelyDR

Consider the topologies $T_1, \ldots, T_7$ in Figure 3. In T_1, there is an undeniable channel from *P* to *Auth*. When the platforms are physical ballot boxes, this can be interpreted as the assumption that sufficiently many witnesses see all ballots in the boxes and observe that they are forwarded and considered in the tallying process. The undeniable channel between *H* and *P* in T_2 could, for example, model that witnesses at each polling station observe voters' attempts to cast their ballot, e.g., by scanning their already encrypted ballot on a voting machine [31]. The trusted *P* in T_3 models, for example, that everyone trusts the voting machines used to compute and cast ballots. In this case, the machines can store a trustworthy record of what ballots have been cast for dispute resolution.

In topology T_4, the paths from *H* to *Auth* as well as from *Auth* to *H* are reliable. *P* and *Auth* are respectively trusted to forward and reply. In a remote setting, the reliable channel from H to P could model that voters can always successfully enter messages on their platforms, for example on a working keyboard. The voters could try with several platforms [37] to cast their ballot remotely and receive a confirmation from *Auth* or, in the worst case, go to a physical polling station to do so. The assumptions then model that the voters can find a working platform and website (e.g., public platforms in libraries, polling places, etc.) or they can find a polling station that issues them with a valid confirmation before the election closes. In T_5 and T_6, the undeniable channels could model a distributed ledger on which everyone can respectively observe when confirmations are issued or ballots are cast. Finally, T_7 could model a remote setting similar to T_4, but where ballots are cast by the trusted platforms.

B. Resolving dispute D2 in protocols without re-voting

In practice, the properties *VoterC*(*Auth*) and *TimelyP*(*Auth*) can be established in a protocol that provides evidence that a ballot was received by *Auth*. For example, this can be achieved by an undeniable channel or by a confirmation that is sent back from *Auth* to the voter upon a ballot's receipt. *Faulty*(*Auth*, b) can then be defined as the set of traces where a ballot b was received by *Auth* but is not in the set of recorded ballots on the bulletin board (see Section VI-D for a concrete example). In contrast, it is often unclear how *VoterA*(*Auth*) can be established as a voter who abstains cannot prove the *absence* of a message. To solve this issue, we show that *VoterA*(*Auth*) is, in many cases, entailed by the *Uniqueness* property, defined next, that can be achieved using standard techniques. We prove this for protocols without re-voting and assume such protocols in the rest of this section.

Uniqueness(*Auth*) states that whenever any recorded ballots are published and *Auth* is not considered dishonest according to *Faulty*(*Auth*, b) for some $b \neq \perp$, then each recorded ballot b' has been sent by a unique eligible voter H for which *castBy*(b') $= H$. Thereby, the ballot can be sent as part of a larger composed message. To express that a message m' is a subterm of another message m, we write $m \vdash m'$. As everyone can evaluate *Faulty*(*Auth*, b), the property's preconditions and thus *Uniqueness*(*Auth*) are verifiable by everyone.

Definition 7. Let the length of the list $[b]$ be n.

$$\begin{aligned}
&\textit{Uniqueness}(\textit{Auth}) := \{tr \mid b \neq \perp \wedge tr \notin \textit{Faulty}(\textit{Auth}, b) \\
&\wedge \textit{BB}_{rec}([b]) \in tr \wedge j \in \{1,\dots,n\} \wedge i \in \{1,\dots,n\} \implies \\
&\qquad \exists [H], i', j', A_1, A_2, m_1, m_2.\ \textit{BB}_H([H]) \in tr \\
&\qquad \wedge \textit{castBy}([b]_i) = [H]_{i'} \wedge \textit{castBy}([b]_j) = [H]_{j'} \\
&\quad \wedge \textit{send}([H]_{i'}, A_1, m_1) \in tr \wedge \textit{send}([H]_{j'}, A_2, m_2) \in tr \\
&\quad \wedge m_1 \vdash [b]_i \wedge m_2 \vdash [b]_j \wedge (i \neq j \implies [H]_{i'} \neq [H]_{j'})\}.
\end{aligned}$$

The property's guarantees are similar to *eligibility verifiability* [24] in that both state that each element of a list on the bulletin board is associated with a unique eligible voter and we compare the two notions in more detail in [6, Appendix C3]. Note that the property can only hold for protocols where the list of eligible voters is publicly known.

Intuitively, if a protocol satisfies *Uniqueness*(*Auth*), then a ballot recorded for the voter H implies that H cast it. Thus, for any ballot that was not cast by H, *Auth* cannot convincingly claim the contrary and an honest voter is thus protected in disputes *D2*. In particular, the traces in the protocol also satisfy *VoterA*(*Auth*). We prove the following theorem in [6, Appendix B].

Theorem 2. Let *Pr* be a protocol in our class without re-voting and where a voter who abstains does not send any message and let T be a topology in our class.

$$\begin{aligned}
&\forall tr \in \textit{TR}(\textit{Pr}, T).\\
&tr \in \textit{Uniqueness}(\textit{Auth}) \implies tr \in \textit{VoterA}(\textit{Auth}).
\end{aligned}$$

The theorem has the practical application that, while it is often unclear how *VoterA*(*Auth*) can be directly realized, *Uniqueness*(*Auth*) can easily be achieved using standard techniques, such as voters signing their ballots. We provide an example in Section VI-D.

C. How to use our formalism

Given the above results, our formalism can be used to analyze whether a protocol *Pr* and topology T in our class of voting protocols satisfy all dispute resolution properties introduced in Section IV. If there is no topology $T_1, \dots, T_7$ in Figure 3, such that $T_i \sqsubseteq T$, then we can immediately conclude, by Theorem 1, that *TimelyP*(*Auth*) and *AuthP*(*Auth*) cannot hold while the protocol is also functional. Otherwise, analysis is required whether the properties are indeed satisfied by *Pr*.

First, let *Pr* be a protocol that defines a dispute resolution procedure, i.e., specifies the set *Faulty*. Our formalism is mainly intended for this case and can directly be used to analyze whether *VoterC*(*Auth*), *TimelyP*(*Auth*), *VoterA*(*Auth*), and *AuthP*(*Auth*) hold in such a protocol. In protocols without re-voting that satisfy *Uniqueness*(*Auth*) and the preconditions of Theorem 2, *VoterA*(*Auth*) can also be proven by proving *Uniqueness*(*Auth*) and concluding *VoterA*(*Auth*) by Theorem 2.

If a protocol *Pr* does not define a dispute resolution procedure *Faulty* then our properties are also undefined. Nevertheless, one can still try to define a verdict *Faulty* using the protocol's specified signals *BB*, *Ev*, and *Pub* and the terms contained in these signals. Our formalism can then be used to establish for each such *Faulty* which properties are satisfied. However, to prove that no definition of *Faulty* achieves dispute resolution, all possible combinations and relations of the above signals and their terms must be considered. Thus, it is in general not straightforward to efficiently conclude that no appropriate definition of *Faulty* exists for a given protocol.

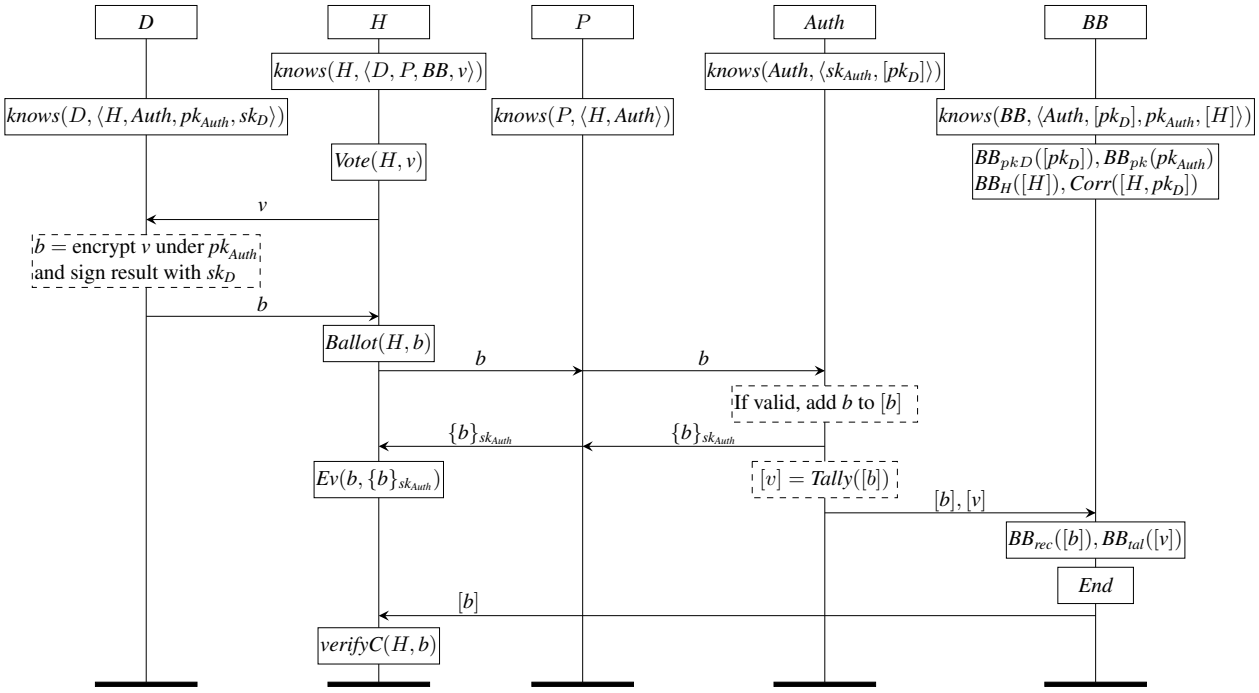
D. A mixnet-based voting protocol with dispute resolution

To demonstrate the applicability of our formalism, we next analyze *MixVote*, a standard mixnet-based voting protocol inspired by [5] with a dispute resolution procedure similar to [28]. In particular, we show how *Faulty* is instantiated, how the properties *VoterC*(*Auth*) and *TimelyP*(*Auth*) differ in practice, and that our dispute resolution properties are compatible with standard voting properties, such as verifiability and receipt-freeness. Due to space constraints, we only describe the protocol's main features here, omitting some details such as the auditor's role and the precise definition of some functions and equations in the term algebra. For the detailed protocol specification, the properties' formal definitions, and the proofs we refer to [6, Appendix C].

1) Topology: We consider a topology T_{MV} that is as T_4 in Figure 3d, except that there is also a trusted off-line device D, which is connected to the voter H by (bidirectional) secure, default channels. T_{MV} specifies reliable channels between H and the platform P and between P and the authority *Auth*. Also, P and *Auth* are partially trusted to forward messages and reply to messages, respectively. Thus, by Theorem 1, it is possible to achieve *TimelyDR* in the topology T_{MV}.

2) Protocol: We present the protocol as a *message sequence chart*, where each role is depicted by a vertical *life line* and where the box on top names the role. A role's life line denotes the role's events, ordered sequentially. A role's sent and received messages are depicted on top of arrows that start at the sender and end at the recipient. Also, we denote explicit signals by solid squares and the roles' internal computations by dashed squares.

MixVote's simplified specification is depicted in Figure 4. The protocol's setup specifies that at each point in time, only one election takes place (i.e., there are no parallel sessions) and each voter possesses a unique trusted device D to which he has exclusive access. All devices are equipped with a unique signing key sk_D and the authority with a unique secret key sk_{Auth}. The corresponding verification keys from the devices $[pk_D]$ are known to *Auth* and *Auth*'s public key pk_{Auth} is known



For $b = \bot$: $\mathit{Faulty}(\mathit{Auth}, b) := \{\}$.

For $b \neq \bot$: $\mathit{Faulty}(\mathit{Auth}, b) := \{tr | (\exists [b], pk_{Auth}, c.\ \mathit{BB}_{pk}(pk_{Auth}) \in tr \wedge \mathit{Ev}(b, c) \in tr \wedge \mathit{ver}(c, pk_{Auth}) = b \wedge \mathit{BB}_{rec}([b]) \in tr \wedge b \notin [b])$
$\vee (\exists [b], [pk_D].\ \mathit{BB}_{rec}([b]) \in tr \wedge \mathit{BB}_{pkD}([pk_D]) \in tr \wedge$ *not all ballots in* $[b]$ *contain a signature associated with a unique key in* $[pk_D]) \}$

Fig. 4: Simplified protocol specification for *MixVote*, without the auditor role and the full function definitions. Here $pk_D = pk(sk_D)$, $pk_{Auth} = pk(sk_{Auth})$, and $\mathit{castBy}(b) = H$ holds iff $\exists pk.\ \mathit{ver}(b, pk) \neq \bot \wedge \langle H, pk \rangle \in [H, pk] \wedge \mathit{Corr}([H, pk]) \in tr$. The protocol's setup specifies a single agent *Auth*, that each voter H is associated with a unique trusted off-line device D, and that there is no restriction on the relation between voters H and platforms P. The role for a voter H who abstains consists of receiving the list of recorded ballots from the bulletin board followed by the signal $\mathit{verifyA}(H, \emptyset)$.

to all devices. Moreover, all these public keys are published on *BB* (denoted by the signals BB_{pkD} and BB_{pk}, respectively). Additionally, at the protocol's start *BB* knows and publishes the list of eligible voters $[H]$ (denoted by the signal BB_H) and which verification key corresponds to which voter. The latter is denoted by the signal $\mathit{Corr}([H, pk_D])$, where each pair $\langle H, pk_D \rangle$ in the list denotes that the signing key corresponding to pk_D is installed on H's device.

To vote, a voter H uses his device D to compute the ballot as follows: the vote is encrypted under *Auth*'s public key and signed by the device. Then, the voter casts his ballot by entering it on any platform P, which forwards it over the network to *Auth*. For each received ballot b, *Auth* checks b's validity, namely whether b contains a signature corresponding to an eligible voter who has not previously voted. If this is the case, *Auth* adds b to the list of recorded ballots $[b]$. Moreover, as in other protocols [28], to achieve dispute resolution, *Auth* sends back a confirmation to the voter H. The confirmation consists of H's ballot b signed by *Auth* and serves as evidence that b was indeed received by the authority. The voter keeps this confirmation as evidence for later disputes (indicated by the signal *Ev*).

After the voting phase, *Auth* computes the tally from the recorded ballots $[b]$. For this, a standard mixnet is used to decrypt the ballots. This procedure has the properties that no one can learn the correspondence between the encrypted ballots and the decrypted votes. Nevertheless, the mixnet produces evidence, which is published by *Auth* on the bulletin board, that allows everyone to verify that the tally was computed correctly. We describe the detailed functions and equations modeling the *Tally* function in [6, Appendix C1]. Also, we describe there the detailed information that is produced by the mixnet and published on *BB* and how an auditor inspects this information to verify the tally.

Among other information, *Auth* publishes on *BB* the recorded ballots $[b]$ and the votes in the final tally $[v]$, as shown in Figure 4. This allows a voter to read the recorded ballots on *BB* and verify that his ballot is included in this list.

A voter who abstains does not send any messages. After the results are published, he reads the list of recorded ballots $[b]$ on *BB* and believes at that step that no ballot should be recorded for him, which is denoted by the signal $verifyA(H, \emptyset)$.

We complete the protocol's specification with the definitions of the function *castBy* and the dispute resolution procedure *Faulty*. *castBy* specifies that a ballot b is considered to be cast by the voter H if the ballot's signature can be verified with the verification key that is associated with H. *Faulty* specifies that *Auth* is considered dishonest in all traces where (a) some agent possesses evidence consisting of a ballot b signed by *Auth* that is not included in the recorded ballots $[b]$ on the bulletin board or (b) not all published recorded ballots $[b]$ contain a signature of a unique eligible voter. *castBy* is defined in Figure 4's caption and the description of *Faulty* is given in Figure 4, although we omit here the details of how we model (b).

3) Dispute resolution: Intuitively, by the channel and trust assumptions, each voter who casts a ballot b receives, before the election's end, a confirmation. As this confirmation serves as evidence that b must be on *BB*, *VoterC*(*Auth*) and *TimelyP*(*Auth*) hold. Furthermore, since no one can forge *Auth*'s signature, for a ballot b that was not actually received by *Auth* no one can produce (false) evidence that b should be on *BB*. Thus, *Auth* cannot be falsely convicted and *AuthP*(*Auth*) holds too. Moreover, *Uniqueness*(*Auth*) holds because, when *Faulty* does not hold in an execution, all recorded ballots are signed (and thus were sent) by a unique eligible voter. In particular, *Uniqueness*(*Auth*) implies *VoterA*(*Auth*) in *MixVote*.

To understand the difference between *VoterC*(*Auth*) and *TimelyP*(*Auth*), take a topology T'_{MV} equal to T_{MV} except that *Auth* is untrusted. Assume for simplicity that a voter can interpret whether *Auth*'s signature on the confirmation is valid. In reality, this would require an additional protocol step where the voter uses a device. When the protocol is run in T'_{MV}, it satisfies *VoterC*(*Auth*), as a voter only proceeds with his verifiability check when he has previously received a valid confirmation that convinces everyone that his ballot must be recorded. However, *TimelyP*(*Auth*) is violated as *Auth* may never reply with a valid confirmation and thus block a voter. Consequently, there is an unresolved dispute where an outside observer cannot tell whether a voter did not cast a ballot or the authority did not send a confirmation. In contrast, when the protocol is run in topology T_{MV}, *Auth* always sends a timely response and such disputes do not occur.

4) Standard voting properties: In addition to the dispute resolution properties, we prove in [6, Appendix C] that *MixVote* satisfies end-to-end verifiability, consisting of individual verifiability and *tallied-as-recorded*, as well as *eligibility verifiability* [24]. Tallied-as-recorded and eligibility verifiability are two universal verifiability properties that respectively denote that an auditor can verify that the recorded ballots are correctly counted in the final tally and that each vote in the final tally was cast by a unique eligible voter. We also prove that *MixVote* satisfies *receipt-freeness* [18], which denotes that a voter cannot prove to the adversary how he voted, even when he provides the adversary with all secrets that he knows. Intuitively, receipt-freeness holds because the adversary cannot access the voter's device D. Moreover, the evidence used for disputes only contains the ballot and does not reveal the underlying (encrypted) vote.

5) Proofs: We prove in [6, Appendix C4] and by the Tamarin files in [33] that *MixVote* satisfies all above mentioned properties when run in the topology T_{MV}. In particular, we establish most of the properties by automatically proving them for one voter who casts a ballot in Tamarin and by proving them for an arbitrary number of voters by pen-and-paper proofs. The only exceptions are: receipt-freeness, which we prove by Tamarin's built in support for observational equivalence [3]; *VoterA*(*Auth*), which we deduce (by hand) from *Uniqueness*(*Auth*) using Theorem 2; and end-to-end verifiability which we deduce (by hand) from individual verifiability and tallied-as-recorded.

VII. Related work

A. Dispute resolution in poll-site voting protocols

The idea of dispute resolution has been informally considered for *poll-site* voting protocols. In [17], the property considered is called *non-repudiation* and requires that failures *"can not only be detected, but (in most cases) demonstrated"* and that no false convictions can be made. [2] informally considers the properties *contestability* and *defensibility*, which are similar to our dispute resolution properties in that they also protect the honest voters and the honest authority. Contestability requires that some guarantees hold for a voter when he starts the voting process at a polling station. In contrast, our properties *TimelyP*(*Auth*) and *VoterC*(*Auth*) are also suitable for remote settings and respectively make guarantees once a voter casts his ballot and believes that it should be recorded. Moreover, [2]'s definitions are informal and they do not consider timeliness.

In most poll-site voting protocols that consider dispute resolution, voters receive a confirmation as evidence that their ballot was accepted by the authority [8], [11], [12], [14], [17], [22], [36]. In some protocols [11], [17], this confirmation contains the authority's digital signature. In the protocols based on Scantegrity [12], [14], [22], [36] the confirmation consists of a code that is (physically) hidden on the ballot by invisible ink and revealed when a voter marks his choice. A voter's knowledge of a valid code serves as evidence that he voted for a candidate. Thus, when a wrong ballot is recorded, a voter can prove the authority's dishonesty by revealing the code.

Compared to remote voting settings, poll-site protocols profit from the fact that on-site witnesses can observe certain actions. For example, if voters are repeatedly prevented from casting their ballots, this is visible to other voters and auditors in the polling station. Some protocols [22] even explicitly state that voters should publicly declare some decisions before entering them on the voting machine to avoid disputes regarding whether the voting machine correctly followed their instructions. Our notion of undeniable channels allows one to formally consider such assumptions during protocol analysis.

B. Dispute resolution in remote voting protocols

Remotegrity [37] is a remote voting protocol based on Scantegrity, where paper sheets are sent to the voters by postal mail and ballots are cast over the Internet. As with Scantegrity II and III [12], [14], [36], to achieve dispute resolution some codes on these sheets are obscured by a scratch-off surface. If a voter detects a (valid) ballot that is incorrectly recorded for him, he can show to anyone that he has not yet scratched off the relevant codes on his sheets and thus the authority must have falsely recorded this ballot.

[37] discusses several dispute scenarios with respect to whether a ballot is recorded correctly. However, it is stated that *"The [authority] can always force a denial-of-service [..] What Remotegrity does not allow is the [authority] to fully accept (i.e., accept and lock) any ballot the voter did not cast without the voter being able to dispute it."* Thus, the focus is on disputes *D2* in Figure 2, while timeliness in disputes *D1* is not further explored. Moreover, the considered properties as well as the assumed setting are not specified precisely and thus the properties cannot be proven. In contrast, our model enables specifying detailed adversary and system assumptions and provides definitions of dispute resolution properties that can be formally analyzed.

C. Accountability

Our dispute resolution properties are closely related to different notions of *accountability* [10], [25], [26]. Both accountability and our properties formalize how misbehaving protocol participants are identified. While the accountability definitions are generic and allow one to blame different agents in different situations, we focus on understanding what disputes and properties are relevant for voting.

Two accountability definitions have been instantiated for voting protocols. First, accountability due to *Küsters et al.* [26] was instantiated for *Bingo Voting* [9] in [26], for *Helios* [1] in [27], and for *sElect* [28]. These instantiated notions of accountability state that when a defined goal is violated, then some (dishonest) agents can be blamed by a *judge*. A judge may blame multiple parties. As a result, in [27] accountability does not guarantee an unambiguous verdict when a voter claims that his ballot is incorrectly recorded. That is, the property does not guarantee the resolution of such disputes even when the voter is honest. The same holds in [26] and [28] for disputes where a voter claims that he did not receive a required confirmation. To avoid ambiguous verdicts, [26] proposes an alternative accountability property where voters' claims that they did not receive a required confirmation are just ignored. However, this property does not guarantee that the authority is blamed in all situations where an (honest) voter's ballot is not recorded correctly and dispute resolution does not hold.

Second, accountability due to *Bruni et al.* [10] has been instantiated for Bingo Voting in [10]. In this work, *accountability tests* decide whether a given agent should be blamed. However, the accountability test takes as input a ballot and a confirmation that the voter received when casting his ballot. Thus disputes where a voter claims that he cannot receive a confirmation are not considered at all.

In contrast to these two accountability notions, we also consider and resolve disputes where a voter claims that he did not receive a required response from *Auth* after casting the ballot by the property *TimelyP*(*Auth*). Moreover, our topology characterization allows us to quickly assess when given assumptions are insufficient to satisfy *TimelyP*(*Auth*).

D. Other related properties

Collection accountability [7] states that when a vote is incorrectly collected, the voter should be provided with evidence to convince an *"independent party"* that this is the case, but it has neither been formally defined nor analyzed. *Dispute freeness* [32] states that there is never a dispute. This property is considered in voting protocols where voters are modeled as machines that conduct an election by engaging in a multi-party protocol [35], [23] and is thus inappropriate for large scale elections where voters must be assumed to have limited computational capabilities. Finally, the FOO protocol [20] allows voters to claim that something went wrong. However, without additional assumptions, FOO does not satisfy our dispute resolution properties. In particular, the signed ballot a voter receives does not prove that the *counter*, who is responsible for tallying, has received the ballot.

VIII. Conclusion

Dispute resolution is an essential ingredient for trustworthy elections and worthy of a careful, formal treatment. Based on a systematic analysis of disputes, we proposed new dispute resolution properties and introduced timeliness as an important aspect thereof. We fully characterized all topologies that achieve timeliness. This provides a formal account for the intuition that timeliness requires strong assumptions. For example, it is not achievable in standard remote voting settings where a network adversary can simply drop messages.

While we have focused on necessary assumptions for dispute resolution, in real elections there are other properties, notably privacy, which may require other assumptions. As future work, we would like to investigate how our topology hierarchy must be adapted for these properties and to characterize the required assumptions for them. The combination of such results with our characterization could lead to new insights about the possibility of achieving different properties simultaneously. Furthermore, such combined results could be a starting point to identify the topologies enabling all properties required in voting; this would help in election design to quickly assess the minimal required setups.

References

[1] Ben Adida. Helios: Web-based Open-audit Voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association.

[2] Andrew Appel, Richard DeMillo, and Philip Stark. Ballot-Marking Devices (BMDs) Cannot Assure the Will of the Voters. April 21, 2019. Available at SSRN: https://ssrn.com/abstract=3375755 or http://dx.doi.org/10.2139/ssrn.3375755, Accessed: 2019-12-20.

[3] David Basin, Jannik Dreier, and Ralf Sasse. Automated Symbolic Proofs of Observational Equivalence. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1144–1155. ACM.

[4] David A. Basin, Saša Radomirović, and Michael Schläpfer. A Complete Characterization of Secure Human-Server Communication. In *28th IEEE Computer Security Foundations Symposium, CSF 2015*, pages 199–213. IEEE Computer Society, 2015.

[5] David A. Basin, Saša Radomirović, and Lara Schmid. Alethea: A Provably Secure Random Sample Voting Protocol. In *31th IEEE Computer Security Foundations Symposium, CSF 2018*, pages 283–297. IEEE Computer Society, 2018.

[6] David A. Basin, Saša Radomirović, and Lara Schmid. Dispute Resolution in Voting (full version). 2020. Available at:https://arxiv.org, Accessed: 2020-05-11.

[7] Matthew Bernhard, Josh Benaloh, J. Alex Halderman, Ronald L. Rivest, Peter Y. A. Ryan, Philip B. Stark, Vanessa Teague, Poorvi L. Vora, and Dan S. Wallach. Public Evidence from Secret Ballots. In Robert Krimmer, Melanie Volkamer, Nadja Braun Binder, Norbert Kersting, Olivier Pereira, and Carsten Schürmann, editors, *Electronic Voting*, pages 84–109, Cham, 2017. Springer International Publishing.

[8] Jens-Matthias Bohli, Christian Henrich, Carmen Kempka, Jörn Müller-Quade, and Stefan Röhrich. Enhancing Electronic Voting Machines on the Example of Bingo Voting. *IEEE Trans. Information Forensics and Security*, 4(4):745–750, 2009.

[9] Jens-Matthias Bohli, Jörn Müller-Quade, and Stefan Röhrich. Bingo Voting: Secure and Coercion-Free Voting Using a Trusted Random Number Generator. In Ammar Alkassar and Melanie Volkamer, editors, *E-Voting and Identity*, pages 111–124. Springer Berlin Heidelberg, 2007.

[10] Alessandro Bruni, Rosario Giustolisi, and Carsten Schuermann. Automated Analysis of Accountability. In Phong Q. Nguyen and Jianying Zhou, editors, *Information Security*, pages 417–434. Springer International Publishing, 2017.

[11] Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Vanessa Teague, Roland Wen, Zhe Xia, and Sriramkrishnan Srinivasan. Using Prêt à Voter in Victoria State Elections. In *2012 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '12*, 2012.

[12] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, USENIX Security'10, pages 291–306. USENIX Association, 2010.

[13] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1614–1625. ACM, 2016.

[14] D. Chaum, R. T. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, A. T. Sherman, and P. L. Vora. Scantegrity II: End-to-End Verifiability by Voters of Optical Scan Elections Through Confirmation Codes. *IEEE Transactions on Information Forensics and Security*, 4(4):611–627, 2009.

[15] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy (S&P 2008)*, pages 354–368, 2008.

[16] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election Verifiability for Helios under Weaker Trust Assumptions. In *Computer Security - ESORICS 2014*, pages 327–344. Springer International Publishing, 2014.

[17] Chris Culnane, Peter Y. A. Ryan, Steve Schneider, and Vanessa Teague. vVote: a Verifiable Voting System (DRAFT). *CoRR*, abs/1404.6822, 2014.

[18] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-Resistance and Receipt-Freeness in Electronic Voting. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006)*, pages 28–42, 2006.

[19] Danny Dolev and Andrew C. Yao. On the Security of Public Key Protocols. *IEEE Trans. Information Theory*, 29(2):198–207, 1983.

[20] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, ASIACRYPT '92, pages 244–251. Springer-Verlag, 1993.

[21] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter A. Ryan, and Josh Benaloh, editors, *Towards Trustworthy Elections, New Directions in Electronic Voting*, pages 37–63. Springer-Verlag, 2010.

[22] Tyler Kaczmarek, John Wittrock, Richard Carback, Alex Florescu, Jan Rubio, Noel Runyan, Poorvi L. Vora, and Filip Zagórski. Dispute Resolution in Accessible Voting Systems: The Design and Use of Audiotegrity. In *E-Voting and Identify*, pages 127–141. Springer Berlin Heidelberg, 2013.

[23] Aggelos Kiayias and Moti Yung. Self-tallying Elections and Perfect Ballot Secrecy. In *Public Key Cryptography*, pages 141–158. Springer Berlin Heidelberg, 2002.

[24] Steve Kremer, Mark Ryan, and Ben Smyth. Election Verifiability in Electronic Voting Protocols. In *Computer Security – ESORICS 2010*, volume 10, pages 389–404. Springer, 2010.

[25] Robert Künnemann, Ilkan Esiyok, and Michael Backes. Automated Verification of Accountability in Security Protocols. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019*, pages 397–413, 2019.

[26] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and Relationship to Verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010*, pages 526–535, 2010.

[27] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *IEEE Symposium on Security and Privacy, SP 2012*, pages 395–409, 2012.

[28] R. Küsters, J. Müller, E. Scapin, and T. Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *29th IEEE Computer Security Foundations Symposium CSF 2016*, pages 341–354, 2016.

[29] Ueli M. Maurer and Pierre E. Schmid. A Calculus for Secure Channel Establishment in Open Networks. In *European Symposium on Research in Computer Security*, pages 173–192. Springer, 1994.

[30] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In Natasha Sharygina and Helmut Veith, editors, *25th International Conference on Computer Aided Verification (CAV 2013)*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.

[31] Peter YA Ryan, David Bismark, James A Heather, Steve A Schneider, and Zhe Xia. The Prêt à Voter verifiable election system. *IEEE Transactions on Information Forensics and Security*, 4(4):662–673, 2009.

[32] Krishna Sampigethaya and Radha Poovendran. A framework and taxonomy for comparison of electronic voting schemes. *Computers & Security*, 25(2):137–153, 2006.

[33] Lara Schmid. Tamarin input files. https://github.com/tamarin-prover/tamarin-prover/tree/develop/examples/csf20-disputeResolution.

[34] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *25th IEEE Computer Security Foundations Symposium, CSF 2012*, pages 78–94, 2012.

[35] Berry Schoenmakers. A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting. In *CRYPTO*, pages 148–164. Springer-Verlag, 1999.

[36] Alan T. Sherman, Russell A. Fink, Richard Carback, and David Chaum. Scantegrity III: Automatic Trustworthy Receipts, Highlighting over/Under Votes, and Full Voter Verifiability. In *Proceedings of the 2011 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'11, pages 7–7. USENIX Association, 2011.

[37] Filip Zagórski, Richard T. Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. Remotegrity: Design and Use of an End-to-End Verifiable Remote Voting System. In Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, pages 441–457. Springer Berlin Heidelberg, 2013.